Atty. Docket No. MS150965.1

SCHEDULER UI

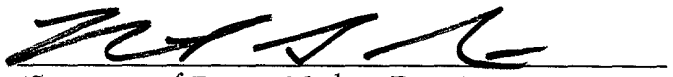
by

Amit Mital, Lucius Gregory Meredith,
Marc Levy, Brian C. Beckman,
Anthony D. Andrews and Terry J. Myerson

CERTIFICATE OF MAILING

I hereby certify that the attached patent application (along with any other paper referred to as being attached or enclosed) is being deposited with the United States Postal Service on this date March 6, 2001, in an envelope as "Express Mail Post Office to Addressee" Mailing Label Number EL782426556US addressed to the: Box Patent Application, Assistant Commissioner for Patents, Washington, D.C. 20231.

Himanshu S. Amin
(Typed or Printed Name of Person Mailing Paper)



(Signature of Person Mailing Paper)

Title: SCHEDULER UI

Technical Field

The present invention relates to computer processes, and more particularly to a system and method for modeling business workflow processes employing a graphical user interface.

Background of the Invention

Transaction processing systems have led the way for many ideas in distributed computing and fault-tolerant computing. For example, transaction processing systems have introduced distributed data for reliability, availability, and performance, and fault tolerant storage and processes, in addition to contributing to a client-server model and remote procedure call for distributed computation. Importantly, transaction processing introduced the concept of transaction ACID properties - atomicity, consistency, isolation and durability that has emerged as a unifying concept for distributed computations. Atomicity refers to a transaction's change to a state of an overall system happening all at once or not at all. Consistency refers to a transaction being a correct transformation of the system state and essentially means that the transaction is a correct program. Although transactions execute concurrently, isolation ensures that transactions appear to execute before or after another transaction because intermediate states of transactions are not visible to other transactions (*e.g.,* locked during execution). Durability refers to once a transaction completes successfully (commits) its activities or its changes to the state become permanent and survive failures.

Many applications for workflow tools are internal to a business or organization. With the advent of networked computers and modems, computer systems at remote locations can now easily communicate with one another. This allows computer system workflow applications to be used between remote facilities within a company. Workflow applications can also be of particular utility in processing business transactions between different companies. Automating such processes can result in significant improvements in efficiency, not otherwise possible. However, this inter-company application of workflow technology requires co-operation of the companies and proper interfacing of the individual company's existing computer systems.

A fundamental concept of workflow analysis is that many business processes can be

interpreted as a sequence of basic transactions called workflows. Workflows have a customer, a performer, and conditions of satisfaction. The customer and performer are roles that participants can take in workflows. In addition, workflows can have observers. In conventional business workflow systems, a transaction comprises a sequence of operations that change recoverable resources and data from one consistent state into another, and if a failure occurs before the transaction reaches normal termination those updates will be rolled back or undone. ACID transactions control concurrency by isolating atomic transitions against each other to create a serializable schedule by delaying updates until committing of transactions. This isolation limits granularity viewed by an observer to the size of the parent transactions, until all child transactions commit within a parent transaction. Therefore, application specific programs cannot be invoked and monitoring of transactions by user's cannot be performed based on any actions occurring within a transaction, until the transaction fails or commits.

The Unified Modeling Language (UML) defines a standard notation for representing flow charts for business workflow processes. However, the UML notation cannot be reduced to a programming language for employing complicated workflow processes. Current business workflow software systems provide scheduling software that requires binding within the scheduling to couple the schedule to real world applications and technologies. Conventional schedules requires code to couple the components of the schedule to application program interface (API) objects and/or server objects for interfacing the schedule with systems within each business or department involved in the business process. These types of schedule software require sophisticated programmers in implementing the software for a given business workflow model. Furthermore, these types of schedule software require modification of each schedule for different technologies.

Accordingly, there is an unmet need in the art for a system and/or method for providing a software tool for modeling a business workflow process that mitigates some of the aforementioned deficiencies associated with conventional software and modeling of business workflow processes.

Summary of the Invention

The present invention relates to a graphical user interface (GUI) scheduler program for modeling business workflow processes. The GUI scheduler program includes tools to allow a user to create a schedule for business workflow processes based on a set of rules defined by the GUI scheduler program. The rules facilitate deadlock not occurring within the schedule. The program provides tools for creating and defining message flows between entities. Additionally, the program provides tools that allow a user to define a binding between the schedule and components, such as COM components, script components, message queues and other workflow schedules. The scheduler program allows a user to define actions and group actions into transactions using simple GUI scheduling tools. The schedule can then be converted to executable code in a variety of forms such as XML, C, C+ and C++. The executable code can then be converted or interpreted for running the schedule.

A user is provided with a GUI schedule interface, which allows the user to create a schedule on a first side of the GUI and to define bindings on the other side of the GUI. During creation of the schedule, the scheduler program prohibits the user creating a schedule that will deadlock the schedule by checking correctness of the schedule flow. In creating the binding, the GUI schedule program provides prompts for specifying interfaces and methods of the components being bound. A data flow connection sheet is provided based on the schedule messages and the binding component interfaces and methods. The data flow of messages is then defined by simply connecting the message ports to binding component interfaces to facilitate proper data flow between entities. It is to be appreciated that the separate data flow sheet could be combined into the GUI scheduler interface or provided via a pop up screen after defining binding component interfaces and methods.

According to one aspect of the invention, a workflow scheduler graphical user interface program is provided. The workflow scheduler graphical user interface program comprises a first screen area adapted to allow a user to create a graphical representation of a business workflow process and a second screen area adapted to allow a user to bind the graphical representation of a business workflow process to at least one technological component.

Another aspect of the invention relates to a business process scheduling program. The business process scheduling program comprises a plurality of schedule tool components

adapted to allow a user to create a representation of a business process schedule according to a set of predefined rules. The scheduling program also comprises a conversion component adapted to convert the schedule created by the user to executable code.

Yet another aspect of the invention relates to a computer readable medium having computer-executable instructions for performing a computer methodology. The methodology comprises the steps of displaying a screen having a first region adapted to allow a user to create a representation of a business workflow process. The screen also has a second region adapted to allow a user to bind the representation of the business workflow process to at least one technological component.

According to a further aspect of the invention, a system is provided. The system facilitates modeling of business processes comprised of a plurality of business operations being representable at a transaction level and an action level. The system comprises a computer-readable medium and a plurality of computer-executable components. The components comprise a graphical user interface and a plurality of modeling components accessible through the graphical user interface. The plurality of modeling components are adapted to allow a user to create a graphical representation of a business process and a binding of the business process to at least one technological component.

Another aspect of the invention relates to a graphical user interface program. The graphical user interface program comprises means for allowing a user to create a graphical representation of a business process and means for allowing a user to create a binding of the graphical representation of the business process to at least one technological component.

To the accomplishment of the foregoing and related ends, the invention then, comprises the features hereinafter fully described and particularly pointed out in the claims. The following description and the annexed drawings set forth in detail certain illustrative aspects of the invention. These aspects are indicative, however, of but a few of the various ways in which the principles of the invention may be employed and the present invention is intended to include all such aspects and their equivalents. Other objects, advantages and novel features of the invention will become apparent from the following detailed description of the invention when considered in conjunction with the drawings.

Brief Description of the Drawings

Fig. 1 illustrates a block diagram of a graphical user interface scheduler program in accordance with one aspect of the present invention.

Fig. 2 illustrates a graphical screen of a graphical user interface scheduler program in accordance with one aspect of the present invention.

Fig. 3 illustrates a menu of business workflow graphical interface components in accordance with one aspect of the present invention.

Fig. 4 illustrates a menu of binding graphical interface components in accordance with one aspect of the present invention.

Fig. 5a illustrates a block diagram of a personal computer system in accordance with an environment of the present invention.

Fig. 5b illustrates a block diagram of an alternate computer system in accordance with an environment of the present invention.

Fig. 6a illustrates a UML interaction diagram of a simplified purchase interaction in accordance with one aspect of the present invention.

Fig. 6b illustrates a modeling interaction diagram of the simplified purchase interaction of Fig. 6a in accordance with one aspect of the present invention.

Fig. 7a illustrates an example of nested roles and transactions in accordance with one aspect of the present invention.

Fig. 7b illustrates an example of valid transaction nesting in accordance with one aspect of the present invention.

Fig. 7c illustrates an alternate example of valid transaction nesting in accordance with one aspect of the present invention.

Fig. 7d illustrates an example of invalid transaction nesting in accordance with one aspect of the present invention.

Fig. 7e illustrates an example of valid role nesting in accordance with one aspect of the present invention.

Fig. 7f illustrates an example of valid role and transaction nesting in accordance with one aspect of the present invention.

Fig. 7g illustrates an alternate example of valid role and transaction nesting in accordance with one aspect of the present invention.

Fig. 8a illustrates an example of a highlighted role in accordance with one aspect of the present invention.

Fig. 8b illustrates an example of a non-highlighted role in accordance with one aspect of the present invention.

Fig. 8c illustrates examples of legal role associations in accordance with one aspect of the present invention.

Fig. 8d illustrates an example of a role before association in accordance with one aspect of the present invention.

Fig. 8e illustrates an example of a role after association in accordance with one aspect of the present invention.

Fig. 8f illustrates an example of an illegal role association in accordance with one aspect of the present invention.

Fig. 9 illustrates an example of possible role connections in accordance with one aspect of the present invention.

Fig. 10 illustrates an example of an associated role with three shapes in accordance with one aspect of the present invention.

Fig. 11a illustrates an example of an action shape attached to an Implementation port in accordance with one aspect of the present invention.

Fig. 11b illustrates an example of a role dropped but not associated in accordance with one aspect of the present invention.

Fig. 11c illustrates an example of a role dropped and associated in accordance with one aspect of the present invention.

Fig. 12a illustrates an example of an associated role prior to port attachment in accordance with one aspect of the present invention.

Fig. 12b illustrates an example of an associated role after port attachment in accordance with one aspect of the present invention.

Fig. 13 illustrates an example of manipulation of actions in an associated role in accordance with one aspect of the present invention.

Fig. 14a illustrates an example of two roles with two communicates in accordance with one aspect of the present invention.

Fig. 14b illustrates an example of a deletion of the second role of Fig. 14a in accordance with one aspect of the present invention.

Fig. 14c illustrates an example of a deletion of the first role of Fig. 14a in accordance with one aspect of the present invention.

Fig. 15 illustrates an example of a (GUI) Transaction Property Editor in accordance with one aspect of the present invention.

Fig. 16a illustrates an example of a (GUI) switch shape when first dropped onto a page in accordance with one aspect of the present invention.

Fig. 16b illustrates an example of the (GUI) switch shape of Fig.16a with three rules added in accordance with one aspect of the present invention.

Fig. 17 illustrates an example of a (GUI) Switch Property Editor in accordance with one aspect of the present invention.

Fig. 18 illustrates an example of a (GUI) Rule Property Editor in accordance with one aspect of the present invention.

Fig. 19a illustrates an example of a bound COM component in accordance with one aspect of the present invention.

Fig. 19b illustrates an example of a bound Script component in accordance with one aspect of the present invention.

Fig. 19c illustrates an example of a bound MSMQ component in accordance with one aspect of the present invention.

Fig. 19d illustrates an example of a bound schedule in accordance with one aspect of the present invention.

Fig. 20a illustrates an example of a (GUI) Method Message Editor for COM components in accordance with one aspect of the present invention.

Fig. 20b illustrates an example of a (GUI) XML Message Editor in accordance with one aspect of the present invention.

Fig. 20c illustrates an example of a (GUI) CALL Message Editor in accordance with one aspect of the present invention.

Fig. 20d illustrates an example of a (GUI) Port References Message Editor in accordance with one aspect of the present invention.

Fig. 21 illustrates an example of a (GUI) Port Properties Editor in accordance with one aspect of the present invention.

Fig. 22 illustrates an example of a simple workflow process in accordance with one aspect of the present invention.

Fig. 23 illustrates an example of a conversion of the simple workflow process of Fig. 22 into XML constructs in accordance with one aspect of the present invention.

Fig. 24 illustrates a modeling scheduling language syntax in extended Backus-Naur Form Notation (EBNF) for mapping the (GUI) scheduling components to a program language in accordance with one aspect of the present invention.

Detailed Description of the Invention

The present invention is now described with reference to the drawings, wherein like reference numerals are used to refer to like elements throughout. The present invention is described with reference to a system and method for modeling a business workflow process employing a graphical user interface (GUI) scheduler program. The GUI scheduler program provides tools for creating a schedule for business workflow process. The tools include constraints defined by a set of rules. The rules mitigate deadlock from occurring within the schedule. The tools allow binding of the schedule to technological components, such as COM components, Script components, Message Queue components and other schedules. The GUI schedule software allows a user to define actions and group actions into transactions. The schedule can then be converted to a programming language for execution.

Fig. 1 illustrates a block diagram of the components in a GUI scheduling software program 10. The GUI scheduling software program 10 includes a business workflow process area 20 and a binding process area 30. The business workflow process area 20 includes a business workflow process 24 created using a plurality of business workflow graphical interface components 22. The binding process area 20 includes a binding process 34 created using a plurality of binding graphical interface components 32. The components in the business workflow process 24 are coupled to the components in the binding process 34 by a plurality of communication ports 25. A data flow connection screen 36 is provided that illustrates the data flow connections associated with variables of binding components and the communications ports allowing a user to view the message flows of the business process.

Fig. 2 illustrates an example of a GUI screen 10' employing the GUI scheduling software of the present invention. The GUI screen 10' includes a business workflow process area 20' and a binding process area 30' separated by a separator bar 26'. The business workflow process area 20' includes a business workflow process 24 consisting of a single action. The single action was generated by clicking on and dragging on an action component in a workflow graphical component menu or stencil 22' into the business workflow process area 20', using a user selection device, for example, a computer mouse. The binding process area 30' includes a single binding component 34' consisting of a COM component. The binding component 34' was created by clicking on and dragging a COM component in a binding graphical component menu or stencil 32' into the binding process area 30', using for example, a computer mouse. The action component 24' is coupled to the COM component via a port 25'. The port 25' is created when the binding component 34' is dragged into the binding process area 30'. A data flow connection screen 36 is provided that illustrates the data flow connections associated with COM component variables and the port 25', allowing a user to view the message flows of the business process to the COM component.

Fig. 3 illustrates an example of components associated with the workflow graphical component menu 22' that may be employed by a user to create a business workflow process model. The workflow graphical component menu 22' includes an action component 40, a transaction component 42, a role component 44, a join component 48, a fork component 50, a begin component 52 and an end component 54. Fig 4 illustrates an example of components associated with the binding graphical component menu 32' that may be employed to bind the business workflow process model to technological components for real world implementation of the business workflow process model. The binding graphical component menu 32' includes a COM component 50, a script component 62, a Message Queue component 64 and a workflow component 66. It is to be appreciated that the above components are merely examples of possible components and a number of additional components may be provided in accordance with the present invention.

One aspect of a system in accordance with the present invention is preferably practiced in the context of a personal computer (PC). A representative hardware environment is depicted in Fig. 5a, which illustrates a typical hardware configuration of a workstation or PC in accordance with a preferred aspect having a central processing unit 70, such as a

microprocessor, and a number of other units interconnected via a system bus 72. The workstation shown in Fig. 5a includes a Random Access Memory (RAM) 74, Read Only Memory (ROM) 76 an I/O adapter 78 for connecting peripheral devices such as disk storage units 80 to the bus 72, a user interface adapter 82 for connecting a keyboard 84, a mouse 86, a speaker 88, a microphone 92, and/or other user interface devices such as a touch screen (not shown) to the bus 72, communication adapter 94 for connecting the workstation to a communication network (*e.g.*, a data processing network) and a display adapter 96 for connecting the bus 72 to a display device 98. The workstation typically has resident thereon an operating system such as Microsoft® Windows® Operating System.

Fig. 5b and the following discussion are intended to provide a brief, general description of a suitable computing environment in which the invention may be executed. While the invention will be described in the general context of computer-executable instructions of a computer program that runs on a server computer, those skilled in the art will recognize that the invention also may be implemented in combination with other program modules. Generally, program modules include routines, programs, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including single- or multiprocessor computer systems, minicomputers, mainframe computers, as well as personal computers, hand-held computing devices, microprocessor-based or programmable consumer electronics, and the like. The illustrated aspect of the invention also is practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. But, some aspects of the invention can be practiced on stand-alone computers. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to Fig. 5b, an exemplary system for implementing the invention includes a conventional server computer 120, including a processing unit 121, a system memory 122, and a system bus 123 that couples various system components including the system memory to the processing unit 121. The processing unit may be any of various commercially available processors, including Intel x86, Pentium and compatible microprocessors from Intel and others, including Cyrix, AMD and Nexgen; Alpha from

Digital; MIPS from MIPS Technology, NEC, IDT, Siemens, and others; and the PowerPC from IBM and Motorola. Dual microprocessors and other multi-processor architectures also can be used as the processing unit 121.

The system bus may be any of several types of bus structure including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of conventional bus architectures such as PCI, VESA, Microchannel, ISA and EISA, to name a few. The system memory includes read only memory (ROM) 124 and random access memory (RAM) 125. A basic input/output system (BIOS), containing the basic routines that help to transfer information between elements within the server computer 120, such as during start-up, is stored in ROM 124.

The server computer 120 further includes a hard disk drive 127, a magnetic disk drive 128, *e.g.*, to read from or write to a removable disk 129, and an optical disk drive 130, *e.g.*, for reading a CD-ROM disk 131 or to read from or write to other optical media. The hard disk drive 127, magnetic disk drive 128, and optical disk drive 130 are connected to the system bus 123 by a hard disk drive interface 132, a magnetic disk drive interface 133, and an optical drive interface 134, respectively. The drives and their associated computer-readable media provide nonvolatile storage of data, data structures, computer-executable instructions, etc. for the server computer 120. Although the description of computer-readable media above refers to a hard disk, a removable magnetic disk and a CD, it should be appreciated by those skilled in the art that other types of media which are readable by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, and the like, may also be used in the exemplary operating environment.

A number of program modules may be stored in the drives and RAM 125, including an operating system 135, one or more application programs 136, other program modules 137, and program data 138. The operating system 135 in the illustrated server computer is the Microsoft Windows NT Server operating system, together with the before mentioned Microsoft transaction Server.

A user may enter commands and information into the server computer 120 through a keyboard 140 and pointing device, such as a mouse 142. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 121 through a serial port

interface 146 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or a universal serial bus (USB). A monitor 147 or other type of display device is also connected to the system bus 123 via an interface, such as a video adapter 148. In addition to the monitor, server computers typically include other peripheral output devices (not shown), such as speakers and printers.

The server computer 120 may operate in a networked environment using logical connections to one or more remote computers, such as a remote client computer 149. The remote computer 149 may be a workstation, a server computer, a router, a peer device or other common network node, and typically includes many or all of the elements described relative to the server computer 120, although only a memory storage device 150 has been illustrated in Fig. 5b. The logical connections depicted in Fig. 5b include a local area network (LAN) 151 and a wide area network (WAN) 152. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment, the server computer 120 is connected to the local network 151 through a network interface or adapter 153. When used in a WAN networking environment, the server computer 120 typically includes a modem 154, or is connected to a communications server on the LAN, or has other means for establishing communications over the wide area network 152, such as the Internet. The modem 154, which may be internal or external, is connected to the system bus 123 via the serial port interface 146. In a networked environment, program modules depicted relative to the server computer 120, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

In accordance with practices of persons skilled in the art of computer programming, the present invention is described below with reference to acts and symbolic representations of operations that are performed by the server computer 120, unless indicated otherwise. Such acts and operations are sometimes referred to as being computer-executed. It will be appreciated that the acts and symbolically represented operations include the manipulation by the processing unit 121 of electrical signals representing data bits which causes a resulting transformation or reduction of the electrical signal representation, and the maintenance of data bits at memory locations in the memory system (including the system memory 122, hard

drive 127, floppy disks 129, and CD-ROM 131) to thereby reconfigure or otherwise alter the computer system's operation, as well as other processing of signals. The memory locations where data bits are maintained are physical locations that have particular electrical, magnetic, or optical properties corresponding to the data bits.

Fig. 6a illustrates a simple purchase interaction diagram in a system with autonomous interacting agents employing a UML notation. The agents (*e.g.*, customer, supplier and shipper) are autonomous in that their activities are concurrent and their lifetimes independent of one another. The interactions between the agents are manifested as exchanges of messages between each other (*e.g.*, purchase orders, purchase order confirmations, ship order). The following example is centered on describing agents in terms of ordering of messages sent and received by the agents and modeling the entire system as a composition of individual agents. A completed purchase in which a product is received by the customer and the product paid for by the customer, represents a completed business workflow process. Fig. 6b illustrates the interaction between customer, supplier and shipper through messages sent and received by the ports. An example of concurrency is illustrated by the invoice receipt action and the ETA receipt action in the customer agent. The GUI scheduler program 10 of the present invention utilizes messages sent and received *via* ports to model business workflow processes.

The present invention will now be described with respect to a rule set associated with the GUI scheduling software 10. A dynamic connector tool connector can be enabled and resides on a toolbar menu and a flow chart stencil. The present invention supports one construct, context, with two applications: roles and transactions. The UI includes one shape for roles and one similar shape for transactions, that each translate to contexts in XML. Figure 7a illustrates the nesting of roles and transactions. The role and transaction shapes share most features, such as Cut/Copy/Paste behavior, wiring control flow into and out-of components, RMA's and association. Role shapes have communicative connections and role ports. Transaction shapes can have compensation and catch pages associated with the transaction. Compensation refers to routines invoked on other transactions upon a failure of a given transaction, while catch refers to routines invoked on a failed transaction. Transactions can only be nested two levels. Figs. 7b and 7c illustrate valid transaction nesting, while Fig. 7d illustrates invalid transaction nesting. There is no limitation on the

nesting of roles. Fig. 7e illustrates valid role nesting. Roles and transactions can be nested within each other as illustrated in Figs. 7f and 7g. The only limitation being that transactions can only be nested two levels.

Fig. 8a illustrates the role shape highlighted, while Fig. 8b illustrates the role shape not highlighted. The role shape has sharp corners, while the transaction shape has rounded corners to differentiate the shape of a role from the shape of a transaction. Once a role is annotated with role ports, it easily differentiated from a transaction. Like the transaction shape, the role shape has only one connection point located at the top center of the shape. The system allows only one incoming control flow to be attached to the role's connection point. This forces the semantics of having only one point on entry into the context of the role. There can be any number of controls flows leaving the role. A control flow leaving a role can be attached to any shape that an action shape can be attached. Control flows that exit a role do so wherever the user desires. The incoming control flow rule is that only one control flow may enter the role. The control flow enters the role at the top center of the role. Maintenance of this rule requires some legal applications of roles and others that will be rejected by the system. Fig. 8c illustrates some examples of "legal" role groupings. Fig. 8d illustrates a role before association and Fig. 8e illustrates a role after association. The role shape can have different changes upon association depending on whether or not there is an incoming control flow. If there is an incoming control flow, then that control flow will have to be re-connected to the role's connection point, and another connector shall be added, attaching the role's connection point to the action state that was originally attached to the incoming control flow. There is no special processing for the control flows that exit a role. Not all role enclosures are legal. Fig. 8f illustrates an illegal role association. There are two control flow lines entering the role. If the user attempts to associate the role with the enclosed shapes, the system will present an error message. This role can't be associated because there is more than one control flow path into the role.

Referring now to Fig. 9, a user is prevented from dragging a control flow control handle from a shape outside of a role to a shape inside of a role, once an action is associated with the role. For example, assuming that the action 1 is already associated with the test role, if a user drags the control flow control handle of action 4 to the connection point B of action 1, the system shall present an error message and delete the created connection. To specify

control flow into a role, a user can connect to the role itself first, and then from the role to the action within the role. The user can drag the control flow control handle of action 4 to the connection point A of the test role in the diagram illustrated in Fig. 9. The user can also drag the control flow control handle of the test role to the connection point B on the action shape because it is one of the shapes that are associated within the test role. A user shall be able to drag the control flow control handle of the role to one of its associated shapes that can accept an incoming control flow. Once drawn, if the control handle is dragged to another shape in the role, this new connection replaces the previous connection (*e.g.*, the role does not fork to many control flows, only one control flow comes out, which could optionally lead to a fork or any other flow chart shape). A user shall be able to drag the control flow control handle of one of the associated shapes to any of the associated shapes or any shape outside of the role, that is not within either a role or transaction. For example, referring to Fig.10, a user can drag the control flow control handle of role 2 to actions 1, 3, 5 only. The user can drag a control flow control handle of action 1 to actions 1, 3, 5 as well as action 2 and 4

Figs. 11a - 11c illustrate communication of a role with an implementation port. If a user drags the control flow control handle of a role that is associated to a shape that is not associated with the role, the system will delete the connection and present an error. The role's control flow will begin with an action associated with the role. The shape will be associated with the role and not with a role or transaction that is nested inside of the role. A role port is added on the outside of the role during the association process for each enclosed action shape that is attached to a port. Figure 11a shows an action shape attached to a port. Figure 11b illustrates a role that includes an action associated with a port where the action is not associated with the role. Fig. 11c illustrates the creation of a role port on the outside of the role during the association process for each enclosed action shape that is attached to a port where the action is associated with the role.

Referring to Fig. 12a illustrating an associated role with three action shapes that are not connected or attached to a port. If a user drags the action-to-port control handle of action 4 to the port 1 / Message 1 connection point, the result will be the diagram of Fig. 12b. A new role port is added to the right hand edge of the port at the same Y location as the action shape. A connector is created between the action shape and the new role port. Another connector is created between the role port and the port on the spacer bar. A user can switch

the role port from the right edge to left edge of the role and vice-versa *via* an RMA on the role port. When a role port is created for connection to a port on a separator bar, the default location of the role port will be on the right side of the role. When a role port is created for connection to another role port, the default location will be the side of the role closest to the other role port. A user will be able to RMA on any point between the action and the destination (port on separator bar or another action in another role) to switch the directionality of the connection from send to receive. If a user drags the action-to-port control handle of action 4 to another port such as port 2 / Message 1, then the system will re-wire the role port-to-port connector to that new port as shown in Fig. 13. Role ports do not have any control handles. To specify or re-specify an action to port connector, the user may drag an action's action to port control handle and drop it on a valid port connection point.

Figs. 14a - 14c illustrate communications between roles. Referring initially to Fig. 14a, a user can drag an action's action to port control handle to an action state that is contained inside of another role. This will create a communicate connection, consisting of two role ports, one on each role, and a communicates message. The default orientation of the communicates message will be horizontal. If a user deletes the action to role port connector, the role port-to-port connector as well as the role port shall be removed. If a user deletes the role port-to-port connector, the action to role port connector will also be removed. If a user deletes the role port, the role port-to-port connector as well as the action-to-role port connector will be removed. If a role is deleted, a user will be prompted to delete just the role shape, or everything within it. If deleting the role with all innards is selected, the GUI scheduler program deletes all of the associated shapes as well as any role ports, action-to-role port connectors, and role port-to-port connectors that are attached. If the deleted role participates in a communicates connector, then a new external port is created that represents the newly deleted role with the name of the deleted role. A new port message is created in the new port for each communicates port message that gets deleted as a result of deleting the role. The role port for each action to port connection will be moved to the right, to be near the new port on the separator bar.

Fig. 14a illustrates two roles communicating. If the user deletes role 2, the system creates a diagram similar to the one illustrated in Fig. 14b. A new port was added to the separator bar that has the same name as the deleted role. Similarly, if the user deleted role 1

from the diagram in Fig. 14a, the system would generate a diagram similar to the one in Fig. 14c. Again the new port is named the same as the deleted role. Furthermore, the Send / Receive value of the port message is obtained from the remaining role, role 2. Table 1 illustrates the properties associated with roles in various states. Role ports have no properties.

TABLE I - ROLE SHAPE PROPERTIES

Shape and shape state	Menu option	Description
Role, Unassociated	1. Edit Properties... 2. Associate the Enclosed actions Within the role	1. Raise property sheet 2. "Group" all shape enclosed within the role shape into the role. Do semantic checks such as only 1 control flow in.
Role, Associated	1. Edit Properties... 2. Disassociate the Enclosed actions Within the role	1. Raise property sheet 2. "Ungroup" all shapes enclosed with the role shape. The control flow connection to the role is extended to the first shape connected within the role.
Role Port	1. Change direction of connection to [sink/source] 2. Move shape to [left/right] of role shape	

Shape Properties

Property Name	Data Type	Description
Name Must be unique amongst roles, transactions, and ports.	String	Unique name to refer to role
Association State (Roles)	Toggle [Associated/Unassociated]	

Transactions have two additional properties that roles lack, that is enable compensation and enable catch. These properties can be enabled independently. Fig. 15 illustrates a pop-up window (GUI) showing transaction properties. If either catch or compensation is enabled, then a new view is created, either "compensation for [transaction-name]" or "catch for [transaction-name]." These views are represented by tabs at the bottom of the page, similar to Business Logic and Data tabs (See Fig. 2). Since transactions can be nested, an interior transaction may have both a catch and a compensation transaction. Those transactions themselves may have both a catch and a compensation transaction. The Business Logic page, which contains root transactions, catch, and compensation pages all look and feel the same, but have completely independent control flows. However, they all share the same separator bar, ports, message, and bindings. Like the main Business Logic page, each catch or compensation page is created only with a begin shape on the flow chart side. Each page can be viewed simultaneously using windowing features. Cut / Copy / Paste is possible between all views (business logic and each compensation or catch page). All workflows will start out with an undeletable "Workflow Catch Page" which is the default error handler for the entire workflow. In this case, there will be three initial pages, Business Logic, Data flow and Workflow Catch. Table 2 illustrates the properties associated with transactions in various states.

TABLE 2 - TRANSACTION SHAPE PROPERTIES

Property Name	Data Type	Description
Name Must be unique amongst roles, transactions, and ports.	String	Unique name to refer to role
Type	Combobox: • Short lived DTC style transaction • Long running compensation based Short lived is default	
Retry count	Integer (default 0)	
Back off time (seconds)	Integer (no default)	
transaction timeout (seconds)	Integer	Enabled only for short lived transactions
Isolation level	Combobox: • Serializable • Read uncommitted • Read committed • Repeatable read Serializable is the default	
Enable Catch	Boolean [Off by Default]	Changing this setting has no visual affect in the Designer. This only changes a setting in the exported XML.

Enable Compensation	Boolean [Off by Default]	Changing this setting has no visual affect in the Designer. This only changes a setting in the exported XML.
[Add/Delete] Catch Sheet	Button	Adds or deletes the Catch sheet. If delete, the user will be prompted "Are you sure you want to delete the Catch sheet for this transaction?"
[Add/Delete] Compensation Sheet	Button	Adds or deletes the Compensation sheet. If delete, the user will be prompted "Are you sure you want to delete the Compensation sheet for this transaction?"

If an action is dropped, no ports or port-messages will be created. If an action is rewired to an existing port a new port message is created that can refer to an existing message or new message depending upon where connection is dropped. A user can connect an action within an associated role (from left or right side) to another action (left or right side) within another (different) associated role. Fields in the new message shape are added and populated from the information from the method (first method selected), if bound to a COM shape. Action to implementation port connectors can be selected and deleted. If any one of (action to role port, role port, role port to communicates message) are deleted, the entire communication connection is deleted (potentially leaving an unused message on the data page). If an action is deleted, any action to port connectors will be deleted, which may trigger a dangling communicates and the creation of a new implementation port.

Fig. 16a illustrates a switch shape that is provided for providing a decision component for the GUI scheduling program. The switch shape can map directly to an XML

programming language component. The switch shape contains rule shapes. Rule shapes are uniquely named. Default rule names will be "Rule x" where x is an integer 1,2, etc, similar to the auto naming of ports, actions, transactions, etc. Multiple switch shapes can contain the same rule. A specific rule will only be used once within a specific switch. Cleanup will delete rules that are no longer used in any switch shapes. Cleanup is a term used to refer to two menu options under the Tools menu. Each option deletes shapes. Each Switch shape always has one, un-editable, "Else" rule shape. When a switch shape is originally dragged onto the page, the only rule will be Default. A user will need to edit properties on the shape to add additional rules, such as the three new rules added to the switch shape illustrated in Fig. 16b. Each switch shape will size itself according to the longest rule that it contains. Each rule will be the same size. The shape has one connection at the top of the diamond where the flow enters the shape. Each rule can have one connection coming out of either the left or right side. A user can wire the rule by dragging the control handle located at the left or right edge of the rule. Redrawing a rule outbound connection, from either side, will not require deletion of the previous connection. The newly drawn control flow connection will replace the old one.

Fig. 17 illustrates an editable pop-up window (GUI) showing switch properties. Switches can be edited to enable a user to add new and existing rules to the switch, remove rules, reorder the rules or edit the rules. "Default" is not part of the picklist available to a user when selecting a rule from the "existing" rules. The "Add" button will create a new rule with a new unique name and add it to the switch. This new rule will be highlighted, so that all the user has to do is click on "Edit" with a pointer. Editing a port message within a port affects everywhere the message/port-message is used and editing a rule within a switch affects everywhere the rule is used. Fig. 18 illustrates an editable pop-up window (GUI) showing rule properties. The rule editor can be raised by selecting the shape on the business logic page and RMA edit properties and by selecting the rule in the switch editor and selecting "Edit". The "Else" rule is not editable and is not displayed in the decision editor. The user can select a switch shape and delete it from the page, similar to any control flow shape. This will not completely delete the rules and therefore cleanup can be used. This is very similar to deleting a port that contains messages. Selecting a rule shape and pressing delete will delete a rule from a switch and any control flow connection out of that rule in the switch. However,

that rule is still available for selection in any other switch. Cleanup can be used to delete unreferenced rules entirely. Rules are not completely deleted even if they no longer have any instances. The rules "cleanup" shall occur when the user cleans up the unused ports and messages. Therefore, the user will not lose any data that they enter without some user specific action.

Fig. 19a illustrates a bound COM component with a method breakout connector. Fig. 19b illustrates a bound Script component, while Fig. 19c illustrates a bound MSMQ component. Fig. 19d illustrates a port bound to another schedule. Ports bound to schedules can send messages. Binding is employed by selecting a component from the binding graphical component menu or stencil 32' and dragging it into the binding process area 30'. When this component is dropped into the binging process area 30', a wizard will start to walk the user through the binding. If the user cancels at any time during the wizard prior to selecting the "Finish" button, the dragged binding shape shall be deleted. The wizards can be started by dropping a binding shape from the menu or stencil and by employing an RMA on an existing binding shape or breakout connector. All states from previous wizard runs will be kept to enable the user to click next to get to the page they want to edit. The first page of every wizard asks the user to specify a port to bind. The user can specify an existing unbound port or enter a new unique port name. If the user specifies binding to a new port, when the wizard is completed, a port shape is added on the separator bar at the same 'Y' value as the binding shape. Port message is not created as a result of any of the binding wizards. Messages are not created on the data page as a result of any of the binding wizards.

The UI provides a different message editor form for each binding technology. There is no enforcement of having one message flow to ports all bound to the same technology. When the user selects a message to edit on the business logic pages, the UI will display the message editor of the port within which the message was selected. If the message flows through ports of different binding types, when the user selects a message to edit on the data page, they will be presented with an option as to which message editor to use. Fig. 20a illustrates a message editor for methods. Fig. 20b illustrates a message editor for XML, while Fig. 20c illustrates a message editor for CALL. Fig. 20d illustrates a Port References Message Editor.

Every message has two system fields ("Result Status", int) and ("Sender", string). The user shall not be able to enter a default value for the "Result Status" or Sender field. In the Field display section of each message editor, Name and Data Type are not editable. Data Types are typically displayed as XML types, regardless of message editor. Field names are unique per message. For the name, each message editor has a combobox containing all existing messages to enable the user to say that this message is exactly another message. Each message editor will have a rename button to the right of the combobox to allow the user to rename the currently selected message. This rename/combox will be the same whether selected from the data or business logic pages. Message names are displayed as a textbox, on both the business logic and data pages. Unique name validation will be done when the user presses the "OK" button. This editor will allow the user to specify two schemas (.XML files). The Message fields frame appears identical to the Method message editor. The names and data types are populated from the top-level XML Elements in the Internal Schema specified in the Message Schema frame. The user will be able to select from all transactions on the page. With COM and Method bindings, the fields of the messages are automatically set to either the [in] or the [out] arguments of method they are pointed to in the port. With MSMQ bindings, the user will have to go into the message editor specify an XML schema, and select nodes in the tree to populate the fields. With CALL bindings, the user will have to go and specify the ports, messages, and transactions being passed into the called schedule.

All data types displayed on the data page will be XML data types. The Data page has one intrinsic message called the port references message that represents all of the ports on the business logic page with a field within the message. Visually, each field will have data type "port". Dynamic ports (those ports specified by the user in the wizard) will have connection points on the left. Field-to-Field connectors can terminate at these fields within the port references shape. All ports will have connection points / control handles on the right. Like any other message on the data page, there can be n number of Field-to-Field connectors starting at any field within the port references message. The port references message shall be updated each time a new port is added, deleted, or edited (name change). The shape shall add, delete or modify the appropriate field shape and its information. The user shall be able to change the relative order of the fields in the port references message for readability. The lines into and out of the port references shape will be visually different from other connectors

on the data page.

Fig. 21 illustrates an editable Port Properties (GUI) pop up screen. Each port will size itself according to the longest port message that it contains. The user shall not be able to manually change the size of a port. Each port message will be the same size. The system has two types of ports, role ports on roles and implementation ports on the separator bar. Role ports can be created by either connecting an action within an associated role to another action within a different associated role or connecting an action contained within a role to a port on the separator bar. Implementation ports can be created in three ways, first by an RMA on the separator bar "Create a new port", by deleting a role that contains role ports or by the binding wizards. Role ports can't be edited. Implementation ports can be edited to enable: changing the port name, enable the user to add port messages to the port, new and existing, to remove port messages, to visually reorder the port messages, to edit port messages or to enable port security. Unidirectional port actions can only send message to ports bound to Workflows. All future connections to that port from actions will default to send. If an existing port was specified in the binding wizard, all current connections will be changed to send (if they are not already). The binding wizard will warn the user that these changes will be made.

The business workflow process models created by the GUI scheduling software can be reduced to a programming language. The models created employing the GUI scheduling software of the present invention will now be illustrated with respect to an example of a business workflow process reduced to a scheduling programming language written in XML (hereinafter referred to as SLANG) including syntax that allows for expression of features associated with the model of the present invention. The programming language allows for users to choose between conventional features of business workflow processes and model specific features in formulating custom models for the user's particular business workflow process. The language is inherently concurrent and allows a user to specify dependency and independence between components, transaction, compensation and checkpoint boundaries, as well as mechanisms for abstracting the workflow from the implementations of the components.

Although, the present example refers to a scheduling language, it is to be appreciated that the present invention can apply to a variety of application programming languages and is not specifically limited to a scheduling language. The scheduling language provides a

mechanism for describing and executing concurrent instances of components. The actions can be mapped to invocations on, for example, common object model (COM) objects, messages in queues, or other native technology behavior. The schedule easily maps onto a distributed environment due to its inherent concurrency, and can further be used to compose other schedules. A schedule can be examined at design time for deadlocks. A compiler can detect deadlock conditions between concurrent actions. A schedule can be stored on a file system, in a database, or embedded in a stream in an application. It is to be appreciated that variation of the business workflow process described herein and the programming language implementing the example would be apparent to those skilled in the art.

Fig. 22 illustrates an example of a simple workflow process including action components or well terminated graphs (WTG) interconnected to one another employing fork and join components. Fig. 23 illustrates the mapping of the simple workflow process to XML tags and a simple program routine that can be compiled and executed to carry out the simple workflow process. Fig. 24 illustrates an example of an XML programming language syntax defined in Extended Backus-Naur Form (EBNF). The syntax includes schedule, ports, messages, contexts and process syntax. Each component and arrangement of components employed in the GUI scheduler program can be mapped to constructs in the XML programming language.

It is to be appreciated that the components in the GUI scheduler program can be mapped to a variety of different programming languages, such as those written in basic, C, C+ or C++. It is also to be appreciated that any programming methodology and/or computer architecture suitable for carrying out the present invention may be employed and are intended to fall within the scope of the hereto appended claims.

The invention has been described with reference to specific aspects. Obviously, modifications and alterations will occur to others upon reading and understanding the foregone detailed description. It is intended that the invention be construed as including all such modifications alterations, and equivalents thereof.